A Project Report

On

**WOMEN SAFETY APPLICATION**

Submitted to

Department of Computer Science

**CHAITANYA (DEEMED TO BE UNIVERSITY)**

In the partial fulfillment of the requirement for the III Year II semester

**B.Sc. ( Cognitive System)**

BY

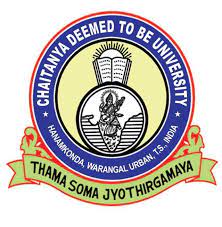
**Sai Vindhya Tenneti (120106008)**

**Prathapagiri SreeVeda (120106031)**

**Md Abu Sufiyan Ahmed (120106051)**

Under the guidance of

**Dr. E. ARAVIND**



DEPARTMENT OF COMPUTER SCIENCE

**CHAITHANYA (DEEMED TO BE UNIVERSITY)**

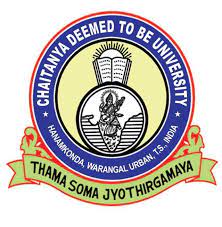
Kishanpura, Hanamkonda, Warangal (T.S)-506001

2022 – 2023

**CHAITHANYA DEEMED TO BE UNIVERSITY**

Kishanpura, Hanamkonda, Warangal (T.S)-506001

**DEPARTMENT OF COMPUTER SCIENCE**



**CERTIFICATE**

This is to certify that **SAI VINDHYA TENNETI** bearing **HTNO:120106008**, **P.SREEVEDA** bearing **HTNO:120106031**, **Md Abu SUFIYAN AHMAD** bearing **HTNO:120106051**, at **Chaitanya (Deemed to be University)** has satisfactorily completed the project entitled **“WOMEN SAFETY APPLICATION”** in the partial fulfillment of the requirements for the award of Degree **B.Sc. (Cognitive Systems)** during the academic Year **2022 – 2023.**

**GUIDE HEAD DEAN ADMINSTRATION**

Mr. E. Aravind Dr. A. Ramesh Babu Dr. S. Kavitha

**Assistant Professor** **Professor** **Professor**

**Internal Examiner External Examiner**

**DECLARATION**

We hereby declare that project report titled **“WOMEN SAFETY APPLICATION”** is an original work done at **Chaitanya (Deemed to be University),** Hanamkonda, Warangal, submitted in the fulfillment for the award of B.Sc (Cognitive Systems) III Year II semester, to the department of Computer Science. We assure you that this project has not been submitted by any degree anywhere in this college or university.

**Sai Vindhya Tenneti**  **(120106008)**

**Prathapagiri SreeVeda**  **(120106031)**

**Md Abu Sufiyan Ahmed**  **(120106051)**

**ACKNOWLEDGEMENT**

We express my sincere gratitude to **Dr.CH. V. Purushotham Reddy, Founder & Chancellor** , **Prof.. B.S.Anuradha**, Dean Faculty of Science, **Prof. S. Kavitha, Dean Administration** of **CHAITANYA (Deemed to be University)** for their unstained interest in providing me facilities to complete the project successfully.

We thank **Dr. A. Ramesh Babu** Professor & Head of the Computer Science, for encouraging us to do such project work and for Providing all the facilities to carry out this Project work.

We extend our special thanks to our guide **Dr. E. Aravind**, Assistant Professor, who helped immensely with her guidance and valuable suggestions during the completion of the project work and who allowed us to do the Project work and gave us a lot of support and constant encouragement throughout this project. We experience from this kind of work is great and will be useful to us in future.

We thank all the faulty members of the Department of Computer Science for sharing their valuable knowledge with us. We extend our thanks to the **Technical Staff** of the department for their valuable suggestions to technical problems. We also thank our family, friends for their hand in bringing this project to this present shape.

**Sai Vindhya Tenneti**  **(120106008)**

**Prathapagiri SreeVeda**  **(120106031)**

**Md Abu Sufiyan Ahmed**  **(120106051)**

Table of Contents

[1. ABSTRACT 1](#_Toc130818887)

[2. INTRODUCTION 2](#_Toc130818888)

[3. PROBLEM DEFINITION 4](#_Toc130818889)

[3.1. EXISTING SYSTEM 5](#_Toc130818890)

[3.2. RELATED WORK 6](#_Toc130818891)

[4. SCOPE OF PROJECT 10](#_Toc130818892)

[5. LITERATURE SURVEY 11](#_Toc130818893)

[5.1. JAVA 11](#_Toc130818894)

[5.2. ANDROID 13](#_Toc130818895)

[5.3. SQLITE 20](#_Toc130818896)

[5.4. ACCELEROMETER SENSORS 36](#_Toc130818897)

[6. SYSTEM REQUIREMENTS 38](#_Toc130818898)

[6.1. SOFTWARE REQUIREMENTS 38](#_Toc130818899)

[6.2. HARDWARE REQUIREMENTS 38](#_Toc130818900)

[6.3. FEASIBILITY STUDY 39](#_Toc130818901)

[7. SYSTEM DESIGN 41](#_Toc130818902)

[7.1. ARCHITECTURE DIAGRAM 41](#_Toc130818903)

[7.2. DATA FLOW DIAGRAMS 42](#_Toc130818904)

[7.3. UNIFIED MODELING LANGUAGE (UML) 43](#_Toc130818905)

[7.4. SEQUENCE DIAGRAM 47](#_Toc130818906)

[7.5. USE CASE DIAGRAM 51](#_Toc130818907)

[7.6. ACTIVITY DIAGRAM 53](#_Toc130818908)

[7.7. CLASS DIAGRAM 56](#_Toc130818909)

[7.8. STATE CHART DIAGRAM 58](#_Toc130818910)

[7.9. DEPLOYEMENT DIAGRAM 59](#_Toc130818911)

[8. FLOWCHART 60](#_Toc130818912)

[9. SAMPLE CODE 61](#_Toc130818913)

[9.1. MainActivity 61](#_Toc130818914)

[9.2. SecondActivity 62](#_Toc130818915)

[9.3. DetailsActivity 68](#_Toc130818916)

[9.4. IncomingCall 74](#_Toc130818917)

[9.5. GPSTracker 79](#_Toc130818918)

[10. SAMPLE SCREEN 85](#_Toc130818919)

[11. FUTURE ENHANCEMENTS 88](#_Toc130818920)

[12. METHOD OF IMPLEMENTATION 89](#_Toc130818921)

[13. TESTING AND VALIDATION 90](#_Toc130818922)

[13.1. TESTING TECHNIQUES 90](#_Toc130818923)

[13.1.1 Unit Testing 90](#_Toc130818924)

[13.1.2 Integration Testing 90](#_Toc130818925)

[13.1.3 Validation Testing 90](#_Toc130818926)

[13.1.4 System Testing 91](#_Toc130818927)

[14. CONCLUSION 92](#_Toc130818928)

[15. BIBILOGRAPHY 93](#_Toc130818929)

[15.1. REFERENCES 93](#_Toc130818930)

[15.2. REFERRED URLS 93](#_Toc130818931)

# **1. ABSTRACT**

Statistics show that approximately 736 million women have been victims of physical and/or sexual intimate violence by a partner or a non-partner. This figure represents one-third of the world's female population. Every 11 minutes a woman or girl was killed at home in 2020 alone (let alone the roads and public areas). This brings the total to 81,000 women per year. 58% of these individuals were killed by a spouse or family member. As a result, there is an undeniable need for safety apps that can assist women in feeling protected and safe on the road.

Women's safety apps, as the name implies, are apps that improve women's safety. People have created a variety of women's safety apps based on their own or their dear ones' experiences. When a user is in trouble, most apps help notify friends and family. Furthermore, the apps can automatically record the incident in order to report the molester in court. Other apps provide information about nearby medical facilities, navigate the safest path, track options, identify crime hotspot areas, and do a variety of other things.

The statistics effectively and concisely convey the necessity for women's security measures. Promoting the women's safety app is one of the simplest and most efficient remedies to the issue, along with law enforcement and defensive training.

The target market for the app is huge enough to attract millions of downloads. But the motivation behind creating women's security applications goes beyond just making money. It is more akin to making a societal contribution to lowering the rates of crime against women.

1. Save the phone numbers of your valued family members in your local storage, which is one of the features we offer in this project.

2. By just shaking the phone three times, you can call and send messages with real-time locations.

3. Purchase women's safety equipment from small business owners.

# **2. INTRODUCTION**

A wide range of tracking systems has been developed so far tracking vehicles and displaying their position on a map, but none of the applications has been developed so far which tracks the mobility of a human being. Now a day’s tracking a person’s mobility has become a crucial issue these days, system which is cost effective and can be used for tracking a human being using a GPS and GPRS equipped mobile phone rather than using a handheld GPS receiver. “The main focus of our project is to reduce the overall cost of tracking based on GPS system which is a satellite-based service which is available 24X7 everywhere in the whole world. GPS system can be used to get location which includes details like latitude, longitude and altitude values along with the timestamp details etc. it a free of cost service available to every individual. In order to track the movement of the person we have used Google Maps for mapping the location sent by the mobile phone. The mobile phone which fetches the GPS location communicates with the server using General Packet Radio Service (GPRS). This service is a low-cost service provided by the service providers which is a wireless data communication system. Mobile phones equipped with GPS receiver are easily available in the market these days and is a booming technology these days. This cell phone technology has enabled us to communicate almost every part of the world across the boundaries. The GSM/GPRS is one of the best and cheapest modes of communication present these days and in future. We are all aware of importance of women's safety, but we must realize that they should be properly protected. Women’s are not as physically strong as men. In an emergency situation a helping hand would be a relief for them. The best way to minimize your chances of becoming a victim of violent crime (robbery, sexual assault, rape, domestic violence) is to identify and call on resources to help you out of dangerous situations. Whether you're in immediate trouble or get separated from friends during a night out and don't know how to get home, having these apps on your phone can reduce your risk and bring assistance when you need it. Here we introduce an app which ensures the safety of women. This helps to identify and call on resources to help the one out of dangerous situations. These reduce risk and bring assistance when we need it and help us to identify the location of the one in danger. This app designed to provide security to women main purpose of this app to provide the awareness on the time of critical situation for women. Generally, you can active this service by clicking on SAVE ME. To do so, you just need to open the app whenever you feel you are in danger and just need to click save me button available in the dashboard. When you click on this button, it opens a new pop-up screen which asks your confirmation and send SMS to those contact which you saved at the time of registration the SMS contain your message and your current location. This app helps you to know about your family, whether they are safe or in trouble.

# **3. PROBLEM DEFINITION**

One of the **fastest growing industries now a day** is mobile industry. There are many competitors in this area who are doing research and development on new platforms & user experience. One such technology is Android from Google which is supported for many manufactured phones. These phones are described as next Generation mobiles [As described by Google]. Nowadays, women have no protection and anything can happen at any time. This app helps the people who are in need of help.

This Medussa app application has a button is called ‘CALL/EMERGENCY’. Opening this app it displays those two buttons just a touch on this help it sends message to the three contact numbers and call one of the contact numbers which are already saved in the database by the user. The android version from 2.2 to 4.4.2 any mobile is applicable for this app.

## **3.1. EXISTING SYSTEM**

In the existing system, the user has to write the message content and select the contact and only then he can send the message but what if the user do not have the that much time or unable to do it.

Recently developed women's safety solutions come in various types such as smartphone apps, security systems and fashionable devices which can be worn every day.

One of the solutions suggests sending a notification to the police or selected contacts when the victim presses the power button. The system then sends the victim's live location after 1 minute. Hence provides a better location when the user or victim moves from one location to another.

Some systems offer a woman authenticated to the device to perform a fingerprint scan. After that, the woman should always scan her fingerprint every minute. Otherwise, the system will send the woman’s location to the registered number via SMS. In the event of a serious situation, the woman does not need to do anything other than simply stop scanning her fingerprints.

## **3.2. RELATED WORK**

A. ABHAYA: AN ANDROID APP FOR THE SAFETY OF WOMEN

This Android application “ABHAYA” for women safety to prevent situations like the Abhaya case in Delhi from happening again. This application uses 3G/2G data connection for tracking the location of the person in trouble and sends a message with the URL of their location from the device to registered contacts. This message gets sent every five minutes to the registered contacts until the “stop” button is clicked. After the application runs whenever the woman presses the “start” button, the application makes a call to the first registered contact and send a message

containing the location URL of the device to all the registered contacts. As it sends location message every five minutes, continuous location tracking of the woman is possible.

B. S-ZONE: A SYSTEM FOR WOMEN SAFETY & SECURITY SYSTEM

In “S-ZONE: A SYSTEM FOR WOMEN SAFETY & SECURITY SYSTEM”, the authors states that “the best way to reduce the chances of becoming a victim of violent crime (robbery, sexual assault, rape, domestic violence) is to identify to help you get out of unsafe situations”. This paper describes the program, S- site designed for android platform for women's safety with the help of the latest developments in mobile technology. This program helps to track the root device via GPS which will help emergency services to rescue a vulnerable person as quickly as possible from harmful situations.

C. SHIELD: PERSONAL SAFETY APPLICATION

“SHIELD: Application for Personal Security” as the name suggests is an application which shields, protects, saves, guards oneself from danger. It sends an instant message with the device’s location to all the registered contacts which helps in live tracking of the location of the woman and provide needed assistance. The main functionality of the system is based on tracking the location. It depends entirely on GPS location tracking and updating on the site in real time. In SHIELD real-time changes in the user area are determined and updated on the website. The update appears on the website within 0.5 seconds depending on the internet connection.

D. WOMEN SAFETY ANDROID APP

In "Women Safety Android App", the authors explain the motto of developing this application is to provide a safe conditions for smartphones as today most people carry smartphones with them wherever they go. It is explained that in the system instead of pressing the SOS on the screen, the victim should press the power button twice to send a notification to the police and to selected contacts even if they do not have internet or GPS connection. And the system will send the continuous location of the victim after one minute i.e. provides a better location, if user or victim relocated from one place to another. It is said to be a key feature of the program is to provide a control panel to the police, so that they have a system where the police and deportees can see any such case or not. When victim press power button then victim’s location will highlight, so police nearby to the victim can easily reach to protect the victim.

E. WOMEN SAFETY MOBILE APP

The working of GPS based “Women Safety Mobile App”. This system starts when the woman authenticated to the device does a fingerprint scan. The woman then needs to constantly scan her finger print every minute, otherwise the system will send the device’s location to the registered number via SMS message and simultaneously sounds a buzzer to alert the nearby people to the situation. The woman does not need to do anything in case of a critical situation except to just stop scanning her fingerprint. The device makes use of GPS, GSM modem, microcontroller based circuit to run the system. This system is very useful in situations where the woman may not be able to press or click the emergency feature. The SMS alert message contains her GPS location and can be sent directly to a few of her registered contacts so that they can provide her with help needed.

**3.3. PROPOSED SYSTEM**

This proposed system contains all the specific capabilities inclusive of live location of tracking and combines all of the functions present in the existing system including GPS tracking and other features which can help in case there is no data connection available. The woman can also use any of the features according to her judgment of the situation faced by her. The purpose of this project is to develop a portable women's safety software tool, which performs the following functions:

1) SOS: SOS sends an alert message to emergency registered contact containing the GPS location of the user every thirty seconds.

2) Siren: A Siren which sounds a loud police siren. This can alert the nearby people of the situation and in some cases may deter the assailant from proceeding with his malicious intentions.

3) Voice Recording: We have also provided a Recording function that records the surrounding sounds which can be used by the victim in case of a police investigation as evidence.

4) Helpline Numbers: The woman can directly call emergency services through the feature Helpline Numbers in the application.

**Modules**

* Module1: Stores a message and 3 contact numbers.
* Module2: Displaying HELP button.
* Module3: Also displays Settings button can set the message and also contact numbers.
* Module4: Save it and then sends the message to 3 contact numbers.
* **Stores a message and 3 contact numbers**: in this module, we must type the message and contact numbers and save it.
* **Displaying help button**: in this module, the app shows a button called help.
* **Displays settings button can set the message and contact numbers**: In this module, there is a settings button, if want to rewrite the message and numbers.
* **Save it and then sends the message to 3 contact numbers**: after setting message and 3 contact numbers then go for the option save. Save will send messages automatically.

# **4. SCOPE OF PROJECT**

The scope of a women safety project typically involves the following aspects:

1. Prevention of violence against women: A women safety project aims to prevent violence against women in all its forms, including physical, emotional, and sexual abuse. This can be achieved through education, awareness campaigns, and advocacy.

2. Support for victims of violence: A women safety project provides support to victims of violence, including access to counselling, medical services, and legal aid.

3. Empowerment of women: A women safety project also focuses on empowering women by providing them with education, training, and resources that help them to protect themselves.

4. Safety in public spaces: A women safety project aims to create safe public spaces where women can move around freely without fear of harassment or violence.

5. Use of technology: Women safety projects can incorporate technology such as mobile apps, GPS tracking, and panic buttons to enhance safety and security.

6. Collaboration with stakeholders: Women safety projects involve collaboration with various stakeholders such as law enforcement agencies, government bodies, NGOs, and local communities to address the issue of violence against women.

Overall, the scope of a women safety project is broad and involves multiple aspects that are interconnected and require a comprehensive approach. The ultimate goal is to create a safe and secure environment for women to live in.

# **5. LITERATURE SURVEY**

## **5.1. JAVA**

Java is an object-oriented language, and is very similar to C++. Java is simplified to eliminate language features that cause common programming errors. Java source code files are compiled into a format called byte code, which can then be executed by a Java interpreter.Features being

* **Platform Independent**

The programs written on one platform can run on any platform provided the platform must have the JVM.

* **Portable**

The feature Write-once-run-anywhere makes the java language portable provided that the system must have interpreter for the JVM.

* **Simple**

Programs are easy to write and debug because java does not use the pointers explicitly. It also has the automatic memory allocation and deallocation system.

* **Multithreaded**

Multithreading means a single program having different threads executing independently at the same time.

* **Robust**

Java has the strong memory allocation and automatic garbage collection mechanism. It provides the powerful exception handling and type checking mechanism as compare to other programming languages.

* **Object Oriented**

To be an Object Oriented language, any language must follow at least the four characteristics.

* Inheritance
* Encapsulation
* Polymorphism
* Dynamic binding
* **Distributed**

The widely used protocols like HTTP and FTP are developed in java. Internet programmers can call functions on these protocols and can get access to the files from any remote machine on the internet rather than writing codes on their local system.

* **Secure**

All the programs in java are run under an area known as the sand box. Security manager determines the accessibility options of a class like reading and writing a file to the local disk.

* **High Performance**

In the beginning interpretation of byte code resulted in slow performance but the advance version of JVM uses the adaptive and just in time compilation technique that improves the performance.

* **Integrated**

Java is an interpreted language as well. Programs run directly from the source code.

## **5.2. ANDROID**

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

**Features:**

**Application framework** enabling reuse and replacement of components.

**Dalvi virtual machine** optimized for mobile devices.

**Integrated browser** based on the open source [WebKit](http://webkit.org/) engine.

**Optimized graphics** powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional).

**SQLite** for structured data storage .

**Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, and GIF) .

**GSM Telephony** (hardware dependent) .

**Bluetooth, EDGE, 3G, and Wi-Fi** (hardware dependent) .

**Camera, GPS, compass, and accelerometer** (hardware dependent).

**Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plug-in for the Eclipse IDE.

**Android application**

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user.

**Libraries**

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:

**System C library** - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices

**Media Libraries** - based on Packet Video’s Open CORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG

**Surface Manager** - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications

**LibWebCore** - a modern web browser engine which powers both the Android browser and an embeddable web view

**SGL** - the underlying 2D graphics engine

**3D libraries** - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer

**Free Type** - bitmap and vector font rendering

**SQLite** - a powerful and lightweight relational database engine available to all applications

**Android Runtime**

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

An Android code editor that helps you write valid XML for your Android manifest and resource files.

It will even export your project into a signed APK, which can be distributed to users.

To begin developing Android applications in the Eclipse IDE with ADT, you first need to download the Eclipse IDE and then download and install the ADT plug-in. To do so, follow the steps given in [Installing the ADT Plugin](http://developer.android.com/sdk/eclipse-adt.html).

**Developing in eclipse with ADT:**

The Android Development Tools (ADT) plug-in for Eclipse adds powerful extensions to the Eclipse integrated development environment. It allows you to create and debug Android applications easier and faster. If you use Eclipse, the ADT plug-in gives you an incredible boost in developing Android applications:

It gives you access to other Android development tools from inside the Eclipse IDE. For example, ADT lets you access the many capabilities of the DDMS tool: take screenshots, manage port-forwarding, set breakpoints, and view thread and process information directly from Eclipse.

It provides a New Project Wizard, which helps you quickly create and set up all of the basic files you'll need for a new Android application.

It automates and simplifies the process of building your Android application.

The Android SDK includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator (based on QEMU), documentation, sample code, and tutorials. Currently supported development platforms include x86-architecture computers running Linux (any modern desktop Linux distribution), Mac OS X 10.4.8 or later, Windows XP or Vista. The officially supported integrated development environment (IDE) is Eclipse (3.2 or later) using the Android Development Tools (ADT) Plug in, though developers may use any text editor to edit Java and XML files then use command line tools to create, build and debug Android application.

**About Native code:**

Libraries written in C and other languages can be compiled to ARM native code and installed, but the Native Development Kit is not yet officially supported by Google. Native classes can be called from Java code running under the Dalvik VM using the System. Load Library call, which is part of the standard Android Java classes.

**Creating an android project:**

The ADT plug-in provides a New Project Wizard that you can use to quickly create a new Android project (or a project from existing code). To create a new project:

* Select File > New > Project.
* Select Android > Android Project, and click Next.
* Select the contents for the project:
* Enter a Project Name. This will be the name of the folder where your project is created.
* Under Contents, select Create new project in workspace. Select your project workspace location.
* Under Target, select an Android target to be used as the project's Build Target. The Build Target specifies which Android platform you'd like your application built against.
* Unless you know that you'll be using new APIs introduced in the latest SDK, you should select a target with the lowest platform version possible, such as Android 1.1.
* Under Properties, fill in all necessary fields :

Enter an Application name. This is the human-readable title for your application — the name that will appear on the Android device.

* Enter a Package name. This is the package namespace (following the same rules as for packages in the Java programming language) where all your source code will reside.
* Select Create Activity (optional, of course, but common) and enter a name for your main Activity class.
* Enter a Min SDK Version. This is an integer that indicates the minimum API Level required to properly run your application. Entering this here automatically sets the min Sdk Version attribute in the [< HYPERLINK "http://developer.android.com/guide/topics/manifest/uses-sdk-element.html"uses-sdk HYPERLINK "http://developer.android.com/guide/topics/manifest/uses-sdk-element.html">](http://developer.android.com/guide/topics/manifest/uses-sdk-element.html) of your Android Manifest file. If you're unsure of the appropriate API Level to use, copy the API Level listed for the Build Target you selected in the Target tab.
* Click Finish.

Once you complete the New Project Wizard, ADT creates the following folders and files in your new project:

src/

Includes your stub Activity Java file. All other Java files for your application go here.

<Android Version>/ (e.g., Android 1.1/)

Includes the android.jar file that your application will build against. This is determined by the build target that you have chosen in the New Project Wizard.

gen/

This contains the Java files generated by ADT, such as your R.java file and interfaces created from AIDL files.

assets/

This is empty. You can use it to store raw asset files. See [Resources and Assets](http://developer.android.com/guide/topics/resources/index.html).

res/

A folder for your application resources, such as drawable files, layout files, string values, etc. See [Resources and Assets](http://developer.android.com/guide/topics/resources/index.html).

AndroidManifest.xml

The Android Manifest for your project. See [The AndroidManifest.xml File](http://developer.android.com/guide/topics/manifest/manifest-intro.html).

Default properties

This file contains project settings, such as the build target. This files is integral to the project, as such, it should be maintained in a Source Revision Control system. It should never be edited manually — to edit project properties, right-click the project folder and select "Properties".

**To create an AVD with the AVD manager:**

* Select Window > Android SDK and AVD Manager, or click the Android SDK and AVD Manager icon (a black device) in the Eclipse toolbar.
* In the Virtual Devices panel, you'll see a list of existing AVDs. Click New to create a new AVD.
* Fill in the details for the AVD.
* Give it a name, a platform target, an SD card image (optional), and a skin (HVGA is default).
* Click Create AVD.

When you first run a project as an Android Application, ADT will automatically create a run configuration. The default run configuration will launch the default project Activity and use automatic target mode for device selection (with no preferred AVD).

**To Create or Modify a Launch Configuration:**

Follow these steps as appropriate for your Eclipse version:

* Open the run configuration manager.
* In Eclipse 3.3 ,select **Run > Open Run Dialog** (or Open Debug Dialog)
* In Eclipse 3.4 (Ganymede), select **Run > Run Configurations** (or Debug Configurations)
* Expand the Android Application item and create a new configuration or open an existing one.

## **5.3. SQLITE**

**SQLite** is an ACID-compliant [embedded](http://en.wikipedia.org/wiki/Embedded_database) [relational database management system](http://en.wikipedia.org/wiki/Relational_database_management_system) contained in a relatively small C programming library. The source code for SQLite is in the public domain.

**Design**

Unlike client-server database management systems, the SQLite engine is not a standalone process with which the application program communicates. Instead, the SQLite [library](http://en.wikipedia.org/wiki/Library_%28computing%29) is linked in and thus becomes an integral part of the application program. The library can also be called dynamically. The application program uses SQLite's functionality through simple [function calls](http://en.wikipedia.org/wiki/Subroutine), which reduces [latency](http://en.wikipedia.org/wiki/Latency_%28engineering%29) in database access as function calls within a single process are more efficient than [inter-process communication](http://en.wikipedia.org/wiki/Inter-process_communication).

An Android application should be fast. Well, it's probably more accurate to say that it should be *efficient*. That is, it should execute as efficiently as possible in the mobile device environment, with its limited computing power and data storage, smaller screen, and constrained battery life.

As you develop your application, keep in mind that, while the application may perform well enough in your emulator, running on your dual-core development computer, it will not perform that well when run a mobile device — even the most powerful mobile device can't match the capabilities of a typical desktop system. For that reason, you should strive to write efficient code, to ensure the best possible performance on a variety of mobile devices.

Generally speaking, writing fast or efficient code means keeping memory allocations to a minimum, writing tight code, and avoiding certain language and programming idioms that can subtly cripple performance. In object-oriented terms, most of this work takes place at the *method* level, on the order of actual lines of code, loops, and so on.

**Introduction:**

There are two basic rules for resource-constrained systems:

Don't do work that you don't need to do.

Don't allocate memory if you can avoid it.

All the tips below follow from these two basic tenets.

Some would argue that much of the advice on this page amounts to "premature optimization." While it's true that micro-optimizations sometimes make it harder to develop efficient data structures and algorithms, on embedded devices like handsets you often simply have no choice. For instance, if you bring your assumptions about VM performance on desktop machines to Android, you're quite likely to write code that exhausts system memory. This will bring your application to a crawl — let alone what it will do to other programs running on the system!

That's why these guidelines are important. Android's success depends on the user experience that your applications provide, and that user experience depends in part on whether your code is responsive and snappy, or slow and aggravating. Since all our applications will run on the same devices, we're all in this together, in a way. Think of this document as like the rules of the road you had to learn when you got your driver's license: things run smoothly when everybody follows them, but when you don't, you get your car smashed up.

Before we get down to brass tacks, a brief observation: nearly all issues described below are valid whether or not the VM features a JIT compiler. If I have two methods that accomplish the same thing, and the interpreted execution of foo() is faster than bar(), then the compiled version of foo() will probably be as fast as or faster than compiled bar(). It is unwise to rely on a compiler to "save" you and make your code fast enough.

**Avoid creating objects:**

Object creation is never free. A generational GC with per-thread allocation pools for temporary objects can make allocation cheaper, but allocating memory is always more expensive than not allocating memory.

If you allocate objects in a user interface loop, you will force a periodic garbage collection, creating little "hiccups" in the user experience.

Thus, you should avoid creating object instances you don't need to. Some examples of things that can help:

When extracting strings from a set of input data, try to return a substring of the original data, instead of creating a copy. You will create a new String object, but it will share the char[] with the data.

If you have a method returning a string, and you know that its result will always be appended to a String Buffer anyway, change your signature and implementation so that the function does the append directly, instead of creating a short-lived temporary object.

A somewhat more radical idea is to slice up multidimensional arrays into parallel single one-dimension arrays:

An array of ints is a much better than an array of Integers, but this also generalizes to the fact that two parallel arrays of ints are also a **lot** more efficient than an array of (int,int) objects. The same goes for any combination of primitive types.

If you need to implement a container that stores tuples of (Foo, Bar) objects, try to remember that two parallel Foo[] and Bar[] arrays are generally much better than a single array of custom (Foo, Bar) objects. (The exception to this, of course, is when you're designing an API for other code to access; in those cases, it's usually better to trade correct API design for a small hit in speed. But in your own internal code, you should try and be as efficient as possible.)

Generally speaking, avoid creating short-term temporary objects if you can. Fewer objects created mean less-frequent garbage collection, which has a direct impact on user experience.

**Use native methods:**

When processing strings, don't hesitate to use specialty methods like String.indexOf(), String.lastIndexOf(), and their cousins. These are typically implemented in C/C++ code that easily runs 10-100x faster than doing the same thing in a Java loop.

The flip side of that advice is that punching through to a native method is more expensive than calling an interpreted method. Don't use native methods for trivial computation, if you can avoid it.

**Prefer Virtual Over Interface:**

Suppose you have a HashMap object. You can declare it as a HashMap or as a generic Map:

Map myMap1 = new HashMap();

HashMap myMap2 = new HashMap();

Which is better?

Conventional wisdom says that you should prefer Map, because it allows you to change the underlying implementation to anything that implements the Map interface. Conventional wisdom is correct for conventional programming, but isn't so great for embedded systems. Calling through an interface reference can take 2x longer than a virtual method call through a concrete reference.

If you have chosen a HashMap because it fits what you're doing, there is little value in calling it a Map. Given the availability of IDEs that refactor your code for you, there's not much value in calling it a Map even if you're not sure where the code is headed. (Again, though, public APIs are an exception: a good API usually trumps small performance concerns.)

**Prefer static over virtual:**

If you don't need to access an object's fields, make your method static. It can be called faster, because it doesn't require a virtual method table indirection. It's also good practice, because you can tell from the method signature that calling the method can't alter the object's state.

**Avoid internal getters/setters:**

In native languages like C++ its common practice to use getters (e.g. i = getCount()) instead of accessing the field directly (i = mCount). This is an excellent habit for C++, because the compiler can usually inline the access, and if you need to restrict or debug field access you can add the code at any time.

On Android, this is a bad idea. Virtual method calls are expensive, much more so than instance field lookups. It's reasonable to follow common object-oriented programming practices and have getters and setters in the public interface, but within a class you should always access fields directly.

**Cache Field Lookup:**

Accessing object fields is much slower than accessing local variables. Instead of writing:

for (int i = 0; i < this.mCount; i++)

dumpItem(this.mItems[i]);

You should write:

int count = this.mCount;

Item[] items = this.mItems;

for (int i = 0; i < count; i++)

dumpItems(items[i]);

(We're using an explicit "this" to make it clear that these are member variables.)

A similar guideline is never call a method in the second clause of a "for" statement. For example, the following code will execute the getCount() method once per iteration, which is a huge waste when you could have simply cached the value as an int:

for (int i = 0; i < this.getCount(); i++)

dumpItems(this.getItem(i));

It's also usually a good idea to create a local variable if you're going to be accessing an instance field more than once. For example:

protected void drawHorizontalScrollBar(Canvas canvas, int width, int height) {

if (isHorizontalScrollBarEnabled()) {

int size = **mScrollBar**.getSize(*false*);

if (size <= 0) {

Size = mScrollBarSize;

}

**mScrollBar**.setBounds(0, *height* - size, width, height);

**mScrollBar**.setParams(

computeHorizontalScrollRange(),

computeHorizontalScrollOffset(),

computeHorizontalScrollExtent(), *false*);

**mScrollBar**.draw(canvas);

}

}

That's four separate lookups of the member field mScrollBar. By caching mScrollBar in a local stack variable, the four member field lookups become four stack variable references, which are much more efficient.

Incidentally, method arguments have the same performance characteristics as local variables.

**Declare Constants Final:**

Consider the following declaration at the top of a class:

static int intVal = 42;

Static String strVal = "Hello, world!";

The compiler generates a class initializer method, called <clinit>, that is executed when the class is first used. The method stores the value 42 into intVal, and extracts a reference from the class file string constant table for strVal. When these values are referenced later on, they are accessed with field lookups.

We can improve matters with the "final" keyword:

static final int intVal = 42;

static final String strVal = "Hello, world!";

The class no longer requires a <clinit> method, because the constants go into classfile static field initializes, which are handled directly by the VM. Code accessing intVal will use the integer value 42 directly, and accesses to strVal will use a relatively inexpensive "string constant" instruction instead of a field lookup.

Declaring a method or class "final" does not confer any immediate performance benefits, but it does allow certain optimizations. For example, if the compiler knows that a "getter" method can't be overridden by a sub-class, it can inline the method call.

You can also declare local variables final. However, this has no definitive performance benefits. For local variables, only use "final" if it makes the code clearer (or you have to, e.g. for use in an anonymous inner class).

**Designing For Responsiveness:**

It's possible to write code that wins every performance test in the world, but still sends users in a fiery rage when they try to use the application. These are the applications that aren't *responsive* enough — the ones that feel sluggish, hang or freeze for significant periods, or take too long to process input.

In Android, the system guards against applications that are insufficiently responsive for a period of time by displaying a dialog to the user, called the Application Not Responding (ANR) dialog. The user can choose to let the application continue, but the user won't appreciate having to act on this dialog every time he or she uses your application. So it's important to design responsiveness into your application, so that the system never has cause to display an ANR to the user.

Generally, the system displays an ANR if an application cannot respond to user input. For example, if an application blocks on some I/O operation (frequently a network access), then the main application thread won't be able to process incoming user input events. After a time, the system concludes that the application has hung, and displays the ANR to give the user the option to kill it.

Similarly, if your application spends too much time building an elaborate in-memory structure, or perhaps computing the next move in a game, the system will conclude that your application has hung. It's always important to make sure these computations are efficient using the techniques above, but even the most efficient code still takes time to run.

In both of these cases, the fix is usually to create a child thread, and do most of your work there. This keeps the main thread (which drives the user interface event loop) running, and prevents the system from concluding your code has frozen. Since such threading usually is accomplished at the class level, you can think of responsiveness as a *class* problem. (Compare this with basic performance, which was described above as a *method*-level concern.)This document discusses how the Android system determines whether an application is not responding and provides guidelines for ensuring that your application is responsive.

**What Triggers ANR:**

In Android, application responsiveness is monitored by the Activity Manager and Window Manager System services. Android will display the ANR dialog for a particular application when it detects one of the following conditions:

No response to an input event (e.g. key press, screen touch) within 5 seconds

A [Broadcast Receiver](http://developer.android.com/reference/android/content/BroadcastReceiver.html) hasn't finished executing within 10 seconds

**How to Avoid ANR:**

Given the above definition for ANR, let's examine why this can occur in Android applications and how best to structure your application to avoid ANR.

Android applications normally run entirely on a single (i.e. main) thread. This means that anything your application is doing in the main thread that takes a long time to complete can trigger the ANR dialog because your application is not giving itself a chance to handle the input event or Intent broadcast.

Therefore any method that runs in the main thread should do as little work as possible. In particular, Activities should do as little as possible to set up in key life-cycle methods such as on Create () and on Resume (). Potentially long running operations such as network or database operations, or computationally expensive calculations such as resizing bitmaps should be done in a child thread (or in the case of databases operations, via an asynchronous request). However, this does not mean that your main thread should block while waiting for the child thread to complete — nor should you call Thread. Wait () or Thread. Sleep (). Instead of blocking while waiting for a child thread to complete, your main thread should provide a [Handler](http://developer.android.com/reference/android/os/Handler.html) for child threads to post back to upon completion. Designing your application in this way will allow your main thread to remain responsive to input and thus avoid ANR dialogs caused by the 5 second input event timeout. These same practices should be followed for any other threads that display UI, as they are also subject to the same timeouts.

The specific constraint on Intent Receiver execution time emphasizes what they were meant to do: small, discrete amounts of work in the background such as saving a setting or registering a Notification. So as with other methods called in the main thread, applications should avoid potentially long-running operations or calculations in Broadcast Receivers. But instead of doing intensive tasks via child threads (as the life of a Broadcast Receiver is short), your application should start a [Service](http://developer.android.com/reference/android/app/Service.html) if a potentially long running action needs to be taken in response to an Intent broadcast. As a side note, you should also avoid starting an Activity from an Intent Receiver, as it will spawn a new screen that will steal focus from whatever application the user is currently has running. If your application has something to show the user in response to an Intent broadcast, it should do so using the [Notification Manager](http://developer.android.com/reference/android/app/NotificationManager.html).

**Reinforcing Responsiveness:**

Generally, 100 to 200ms is the threshold beyond which users will perceive lag (or lack of "snappiness," if you will) in an application. As such, here are some additional tips beyond what you should do to avoid ANR that will help make your application seem responsive to users.

If your application is doing work in the background in response to user input, show that progress is being made (Progress Bar and [Progress Dialog](http://developer.android.com/reference/android/app/ProgressDialog.html) are useful for this).

For games specifically, do calculations for moves in a child thread.

If your application has a time-consuming initial setup phase, consider showing a splash screen or rendering the main view as quickly as possible and filling in the information asynchronously. In either case, you should indicate somehow that progress is being made, lest the user perceive that the application is frozen.

**Designing for Seamlessness:**

Even if your application is fast and responsive, certain design decisions can still cause problems for users — because of unplanned interactions with other applications or dialogs, inadvertent loss of data, unintended blocking, and so on. To avoid these problems, it helps to understand the context in which your applications run and the system interactions that can affect your application. In short, you should strive to develop an application that interacts seamlessly with the system and with other applications.

A common seamlessness problem is when an application's background process — for example, a service or broadcast receiver — pops up a dialog in response to some event. This may seem like harmless behavior, especially when you are building and testing your application in isolation, on the emulator. However, when your application is run on an actual device, your application may not have user focus at the time your background process displays the dialog. So it could end up that your application would display it's dialog behind the active application, or it could take focus from the current application and display the dialog in front of whatever the user was doing (such as dialing a phone call, for example). That behavior would not work for your application or for the user.

To avoid these problems, your application should use the proper system facility for notifying the user — the [Notification](http://developer.android.com/reference/android/app/Notification.html) classes. Using notifications, your application can signal the user that an event has taken place, by displaying an icon in the status bar rather than taking focus and interrupting the user.

Another example of a seamlessness problem is when an activity inadvertently loses state or user data because it doesn't correctly implement the on Pause () and other lifecycle methods. Or, if your application exposes data intended to be used by other applications, you should expose it via a Content Provider, rather than (for example) doing so through a world-readable raw file or database.

What those examples have in common is that they involve cooperating nicely with the system and other applications. The Android system is designed to treat applications as a sort of federation of loosely-coupled components, rather than chunks of black-box code. This allows you as the developer to view the entire system as just an even-larger federation of these components. This benefits you by allowing you to integrate cleanly and seamlessly with other applications, and so you should design your own code to return the favour.

This document discusses common seamlessness problems and how to avoid them. It covers these topics:

**Don’t Drop Data:**

Always keep in mind that Android is a mobile platform. It may seem obvious to say it, but it's important to remember that another Activity (such as the "Incoming Phone Call" app) can pop up over your own Activity at any moment. This will fire the onSaveInstanceState () and on Pause() methods, and will likely result in your application being killed.

If the user was editing data in your application when the other Activity appeared, your application will likely lose that data when your application is killed. Unless, of course, you save the work in progress first. The "Android Way" is to do just that: Android applications that accept or edit input should override the onSaveInstanceState () method and save their state in some appropriate fashion. When the user revisits the application, she should be able to retrieve her data.

A classic example of a good use of this behaviour is a mail application. If the user was composing an email when another Activity started up, the application should save the in-process email as a draft.

**Don’t Expose Raw Data:**

If you wouldn't walk down the street in your underwear, neither should your data. While it's possible to expose certain kinds of application to the world to read, this is usually not the best idea. Exposing raw data requires other applications to understand your data format; if you change that format, you'll break any other applications that aren't similarly updated.

The "Android Way" is to create a Content Provider to expose your data to other applications via a clean, well-thought-out, and maintainable API. Using a Content Provider is much like inserting a Java language interface to split up and componentized two tightly-coupled pieces of code. This means you'll be able to modify the internal format of your data without changing the interface exposed by the Content Provider, and this without affecting other applications.

**Don’t Interrupt The User:**

If the user is running an application (such as the Phone application during a call) it's a pretty safe bet he did it on purpose. That's why you should avoid spawning activities except in direct response to user input from the current Activity.

That is, don't call start Activity () from Broadcast Receivers or Services running in the background. Doing so will interrupt whatever application is currently running, and result in an annoyed user. Perhaps even worse, your Activity may become a "keystroke bandit" and receive some of the input the user was in the middle of providing to the previous Activity. Depending on what your application does, this could be bad news.

Instead of spawning Activity UIs directly from the background, you should instead use the Notification Manager to set Notifications. These will appear in the status bar, and the user can then click on them at his leisure, to see what your application has to show him.

(Note that all this doesn't apply to cases where your own Activity is already in the foreground: in that case, the user expects to see your next Activity in response to input.)

**Go a Lot to Do? Do it in a Thread:**

If your application needs to perform some expensive or long-running computation, you should probably move it to a thread. This will prevent the dreaded "Application Not Responding" dialog from being displayed to the user, with the ultimate result being the fiery demise of your application.

By default, all code in an Activity as well as all its Views run in the same thread. This is the same thread that also handles UI events. For example, when the user presses a key, a key-down event is added to the Activity's main thread's queue. The event handler system needs to dequeuer and handle that event quickly; if it doesn't, the system concludes after a few seconds that the application is hung and offers to kill it for the user.

If you have long-running code, running it inline in your Activity will run it on the event handler thread, effectively blocking the event handler. This will delay input processing, and result in the ANR dialogs. To avoid this, move your computations to a thread. This [Design for Responsiveness](http://developer.android.com/guide/practices/design/responsiveness.html) document discusses how to do that.

**Don’t over load a single activity screen:**

Any application worth using will probably have several different screens. When designing the screens of your UI, be sure to make use of multiple Activity object instances.

Depending on your development background, you may interpret an Activity as similar to something like a Java Applet, in that it is the entry point for your application. However, that's not quite accurate: where an Applet subclass is the single entry point for a Java Applet, an Activity should be thought of as one of potentially several entry points to your application. The only difference between your "main" Activity and any others you might have is that the "main" one just happens to be the only one that expressed an interest in the "android.intent.action.MAIN" action in your Android Manifest.xml file.

So, when designing your application, think of your application as a federation of Activity objects. This will make your code a lot more maintainable in the long run, and as a nice side effect also plays nicely with Android's application history and "back stack" model.

**Extended system themes:**

When it comes to the look-and-feel of the user interface, it's important to blend in nicely. Users are jarred by applications which contrast with the user interface they've come to expect. When designing your UIs, you should try and avoid rolling your own as much as possible. Instead, use a Theme. You can override or extend those parts of the theme that you need to, but at least you're starting from the same UI base as all the other applications. For all the details, read [Applying Styles and Themes](http://developer.android.com/guide/topics/ui/themes.html).

**Design your UI to work with multiple screen resolutions:**

Different Android-powered devices will support different screen resolutions. Some will even be able to change resolutions on the fly, such as by switching to landscape mode. It's important to make sure your layouts and drawables are flexible enough to display properly on a variety of device screens.

Fortunately, this is very easy to do. In brief, what you must do is provide different versions of your artwork (if you use any) for the key resolutions, and then design your layout to accommodate various dimensions. (For example, avoid using hard-coded positions and instead use relative layouts.) If you do that much, the system handles the rest, and your application looks great on any device.

**Assume the network is slow:**

Android devices will come with a variety of network-connectivity options. All will have some data-access provision, though some will be faster than others. The lowest common denominator, however, is GPRS, the non-3G data service for GSM networks. Even 3G-capable devices will spend lots of time on non-3G networks, so slow networks will remain a reality for quite a long time to come.

That's why you should always code your applications to minimize network accesses and bandwidth. You can't assume the network is fast, so you should always plan for it to be slow. If your users happen to be on faster networks, then that's great — their experience will only improve. You want to avoid the inverse case though: applications that are usable some of the time, but frustratingly slow the rest based on where the user is at any given moment are likely to be unpopular.

One potential captcha here is that it's very easy to fall into this trap if you're using the emulator, since the emulator uses your desktop computer's network connection. That's almost guaranteed to be much faster than a cell network, so you'll want to change the settings on the emulator that simulate slower network speeds. You can do this in Eclipse, in the "Emulator Settings" tab of your launch configuration or via a [command-line option](http://developer.android.com/guide/developing/tools/emulator.html) when starting the emulator.

**Don’t assume touch screen or key board:**

Android will support a variety of handset form-factors. That's a fancy way of saying that some Android devices will have full "QWERTY" keyboards, while others will have 40-key, 12-key, or even other key configurations. Similarly, some devices will have touch-screens, but many won't.

When building your applications, keep that in mind. Don't make assumptions about specific keyboard layouts -- unless, of course, you're really interested in restricting your application so that it can only be used on those devices.

**Do converse the device battery:**

A mobile device isn't very mobile if it's constantly plugged into the wall. Mobile devices are battery-powered, and the longer we can make that battery last on a charge, the happier everyone is — especially the user. Two of the biggest consumers of battery power are the processor, and the radio; that's why it's important to write your applications to do as little work as possible, and use the network as infrequently as possible.

Minimizing the amount of processor time your application uses really comes down to [writing efficient code](http://developer.android.com/guide/practices/design/performance.html). To minimize the power drain from using the radio, be sure to handle error conditions gracefully, and only fetch what you need. For example, don't constantly retry a network operation if one failed. If it failed once, it's likely because the user has no reception, so it's probably going to fail again if you try right away; all you'll do is waste battery power.

Users are pretty smart: if your program is power-hungry, you can count on them noticing. The only thing you can be sure of at that point is that your program won't stay installed very long.

**Features**

SQLite implements most of the [SQL-92](http://en.wikipedia.org/wiki/SQL-92) standard for [SQL](http://en.wikipedia.org/wiki/SQL) but it lacks some features. A standalone program called sqlite3 is provided which can be used to create a database, define tables within it, insert and change rows, run queries and manage a SQLite database file. SQLite is a popular choice for local/client SQL storage within a [web browser](http://en.wikipedia.org/wiki/Web_browser) and within the given Hyperlink below:

["http://en.wikipedia.org/wiki/List\_of\_rich\_internet\_application\_frameworks"application framework](rich%20internet%20HYPERLINK%20%22http:/en.wikipedia.org/wiki/List_of_rich_internet_application_frameworks%22application%20framework).

This may be because SQLite's dynamically typed storage matches the [web browser](http://en.wikipedia.org/wiki/Web_browser)'s core languages of [JavaScript](http://en.wikipedia.org/wiki/Javascript) and [XML](http://en.wikipedia.org/wiki/XML). SQLite uses an unusual [type system](http://en.wikipedia.org/wiki/Type_system) for an SQL-compatible DBMS. Instead of assigning a type to a column as in most SQL database systems, types are assigned to individual values; in language terms it is dynamically typed.

## **5.4. ACCELEROMETER SENSORS**

Accelerometers are electronic devices that measure changes in acceleration and orientation of a device. They are widely used in various industries, including aerospace, automotive, robotics, and consumer electronics. In recent years, accelerometers have become increasingly popular in mobile devices, such as smartphones and tablets, due to their ability to detect motion and provide a more immersive user experience.

Accelerometers work by measuring changes in acceleration along three axes, typically labeled as x, y, and z. The most common type of accelerometer is a microelectromechanical system (MEMS) accelerometer, which uses a tiny suspended mass that moves in response to acceleration, producing an electrical signal that can be measured and analyzed. MEMS accelerometers are small, lightweight, and inexpensive, making them ideal for use in mobile devices.

One of the key features of accelerometers is their ability to detect changes in orientation and tilt. This is achieved by measuring changes in the force of gravity acting on the device, which can be used to determine its orientation in space.

Accelerometers can also detect movement and vibration, making them useful for a range of applications, including gaming, fitness tracking, and navigation. Another important feature of accelerometers is their sensitivity and accuracy. Modern accelerometers can detect changes in acceleration as small as a few thousandths of a g (g is the acceleration due to gravity at the Earth's surface), making them highly precise and accurate. This level of sensitivity is critical in applications such as aerospace and automotive, where even small changes in acceleration can have a significant impact on performance and safety. Overall, accelerometers are a powerful and versatile technology that is widely used in a range of industries and applications. Their ability to detect motion, orientation, and acceleration makes them essential components of many modern devices, from mobile phones to airplanes.

**Applications of Accelerometers:**

Accelerometers are used in a wide range of applications, including aerospace, automotive, robotics, and consumer electronics. Some of the most common applications of accelerometers are described below

1. Gaming: Accelerometers are commonly used in mobile gaming applications, allowing users to control games using motion gestures. For example, a user may tilt their device to steer a car in a racing game or shake their device to activate a power-up.

2. Fitness tracking: Many fitness trackers and smartwatches use accelerometers to measure physical activity, such as the number of steps taken or the distance traveled. By detecting changes in acceleration, accelerometers can provide users with valuable insights into their physical activity levels and help them track progress towards fitness goals.

3. Navigation: Accelerometers are often used in navigation applications, such as GPS systems and maps. By detecting changes in orientation and movement, accelerometers can provide users with accurate directions and map data, even in areas with poor GPS reception.

4. Automotive: Accelerometers are widely used in the automotive industry for a range of applications, including crash detection, stability control, and tire pressure monitoring. By detecting changes in acceleration and orientation, accelerometers can provide critical data to help improve vehicle safety and performance.

5. Aerospace: Accelerometers are used extensively in the aerospace industry for a range of applications, including rocket stabilization, spacecraft orientation, and instrument calibration. In space applications, accelerometers are critical for maintaining the correct orientation and alignment of spacecraft and instruments, ensuring accurate data collection and analysis.

Overall, accelerometers are a versatile and powerful technology that is used in a wide range of applications. Their ability to detect motion, orientation, and acceleration makes them essential components of many modern devices, from mobile phones to airplanes, and their sensitivity and accuracy make them critical for applications where even small changes in acceleration can have a significant impact on performance and safety.

# **6. SYSTEM REQUIREMENTS**

## **6.1. SOFTWARE REQUIREMENTS**

* IDE: Eclipse for android developers
* Backend Language: Java
* Frontend language: XML
* Database: SQLite Database

## **6.2. HARDWARE REQUIREMENTS**

1. PROCESSOR : Dual core Processor and above

2. RAM : 1GB and above

3. HARD DISK : 50 GB and above

## **6.3. FEASIBILITY STUDY**

An important outcome of the preliminary investigation is the determination that system requested is feasible. This is to identify the objectives of a new system. Before solving a problem one must know what the problem is. The study is carried out by a small group of people who are familiar with system analysis and design process. Fact finding techniques are used to gather the required information.

The three major areas consider while determining the feasibility of the project are

* Economic Feasibility
* Operational Feasibility
* Technical Feasibility

**Economic Feasibility**:

Economic feasibility attempts to weigh the costs of developing and implementing a new system, against the benefits that would accrue from having the new system in place. This feasibility study gives the top management the economic justification for the new system.

A simple economic analysis which gives the actual comparison of costs and benefits are much more meaningful in this case. These could include increased customer satisfaction, improvement in product quality better decision making timeliness of information, expediting activities, improved accuracy of operations, better documentation and record keeping, faster retrieval of information, better employee morale.

**Operational Feasibility:**

Proposed projects are beneficial only if they can be turned into information systems that will meet the organizations operating requirements. Simply stated, this test of feasibility asks if the system will work when it is developed and installed. Are there major barriers to Implementation? Here are questions that will help test the operational feasibility of a project:

Is there sufficient support for the project from management from users? If the current system is well liked and used to the extent that persons will not be able to see reasons for change, there may be resistance.

Are the current business methods acceptable to the user? If they are not, Users may welcome a change that will bring about a more operational and useful systems.

Have the user been involved in the planning and development of the project?

Early involvement reduces the chances of resistance to the system and in general and increases the likelihood of successful project. Since the proposed system was to help reduce the hardships encountered. In the existing manual system, the new system was considered to be operational feasible.

**Technical Feasibility**

Evaluating the technical feasibility is the trickiest part of a feasibility study. This is because, at this point in time, not too many-detailed design of the system, making it difficult to access issues like performance, costs on (on account of the kind of technology to be deployed) etc.

A number of issues have to be considered while doing a technical analysis.

i) Understand the different technologies involved in the proposed system:

Before commencing the project, we have to be very clear about what are the technologies that are to be required for the development of the new system.

ii) Find out whether the organization currently possesses the required technologies:

Is the required technology available with the organization?

If so is the capacity sufficient?

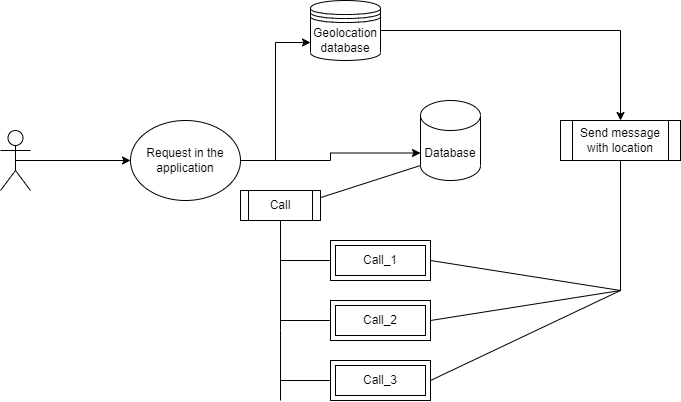
For instance -

“Will the current printer be able to handle the new reports and forms required for the new system?”

# **7. SYSTEM DESIGN**

## **7.1. ARCHITECTURE DIAGRAM**

Architecture diagram is a [diagram](http://en.wikipedia.org/wiki/Diagram) of a [system](http://en.wikipedia.org/wiki/System), in which the principal parts or functions are represented by blocks connected by lines that show the relationships of the blocks. The block diagram is typically used for a higher level, less detailed description aimed more at understanding the overall concepts and less at understanding the details of implementation.



A Call History user for who the application looks like a user interface actually consists of a database called as SQLite that comes along with Android SDK and need no other installation. This is the database that is used to store and retrieve information. This is an application that is developed in java and hence all its features apply here as well such as platform independence, data hiding, portable etc.

## **7.2. DATA FLOW DIAGRAMS**

A **data-flow diagram** (**DFD**) is a graphical representation of the "flow" of data through an [information system](http://en.wikipedia.org/wiki/Information_system). DFDs can also be used for the [visualization](http://en.wikipedia.org/wiki/Data_visualization) of [data processing](http://en.wikipedia.org/wiki/Data_processing) (structured design).

On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process. The idea behind the explosion of a process into more process is that understanding at one level of detail is exploded into greater detail at the next level. This is done further explosion is necessary and an adequate amount of detail is described for analyst to understand the process.

`A DFD is also known as a “bubble chart” has the purpose of clarifying system requirements and identifying major transformations that will become programs in system design.

So it is the starting point of the design to the lowest level of detail. A DFD consists of a series of bubbles joined by data flows in the system.

**7.2.1 DFD Symbols**

In the DFD, there are four symbols

* A Square defines a source or destination of system data.
* An arrow identifies data flow. It is the pipeline through which the information flows.
* A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.
* An open rectangle is a data store ,data at rest or a temporary repository of data

## **7.3. UNIFIED MODELING LANGUAGE (UML)**

**7.3.1. UML Concepts:**

The Unified Modelling Language (UML) is a standard language for writing software blue prints. The UML is a language for

* Visualizing
* Specifying
* Constructing
* Documenting the artefacts of a software intensive system.

The UML is a language which provides vocabulary and the rules for combining words in that vocabulary for the purpose of communication. A modelling language is a language whose vocabulary and the rules focus on the conceptual and physical representation of a system. Modelling yields an understanding of a system.

**7.3.2. Building Blocks of the UML:**

The vocabulary of the UML encompasses three kinds of building blocks:

* Things
* Relationships
* Diagrams

Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

**7.3.3. Things in the UML:**

There are four kinds of things in the UML:

* Structural things
* Behavioural things
* Grouping things
* Anotational things

**Structural things** are the nouns of UML models. The structural things used in the project design are:

First, a **class** is a description of a set of objects that share the same attributes, operations, relationships and semantics.

|  |
| --- |
| Window |
| origin  size |
| open()  close()  move()  display() |

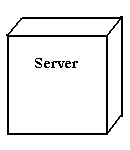
**Fig: Classes**

Second, a **use case** is a description of set of sequence of actions that a system performs that yields an observable result of value to particular actor.



**Fig: Use Cases**

Third, a node is a physical element that exists at runtime and represents a computational resource, generally having at least some memory and often processing capability.



**Fig: Nodes**

**Behavioural things** are the dynamic parts of UML models. The behavioural thing used is:

**Interaction:**

An interaction is a behaviour that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. An interaction involves a number of other elements, including messages, action sequences (the behaviour invoked by a message, and links (the connection between objects).



**Fig: Messages**

**7.3.4. Relationships in the UML:**

There are four kinds of relationships in the UML:

* Dependency
* Association
* Generalization
* Realization

A **dependency** is a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing (the dependent thing).



**Fig: Dependencies**

An **association** is a structural relationship that describes a set links, a link being a connection among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts.



**Fig: Association**

A **generalization** is a specialization/ generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent).



**Fig: Generalization**

A **realization** is a semantic relationship between classifiers, where in one classifier specifies a contract that another classifier guarantees to carry out.



**Fig: Realization**

## **7.4. SEQUENCE DIAGRAM**

UML sequence diagrams are used to represent the flow of messages, events and actions between the objects or components of a system. Time is represented in the vertical direction showing the sequence of interactions of the header elements, which are displayed horizontally at the top of the diagram.

Sequence Diagrams are used primarily to design, document and validate the architecture, interfaces and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario. UML sequence diagrams are useful design tools because they provide a dynamic view of the system behaviour which can be difficult to extract from static diagrams or specifications.

**Actor**

Represents an external person or entity that interacts with the system



**Object**

Represents an object in the system or one of its components



**Unit**

Represents a subsystem, component, unit, or other logical entity in the system (may or may not be implemented by objects)



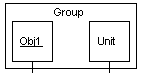
**Separator**

Represents an interface or boundary between subsystems, components or units (e.g., air interface, Internet, network)



**Group**

Groups related header elements into subsystems or components



**Sequence Diagram Body Elements**

**Action**

Represents an action taken by an actor, object or unit



**Asynchronous Message**

An asynchronous message between header elements



**Block**

A block representing a loop or conditional for a particular header element



**Call Message**

A call (procedure) message between header elements



**Create Message**

A "create" message that creates a header element (represented by lifeline going from dashed to solid pattern)



**Diagram Link**

Represents a portion of a diagram being treated as a functional block. Similar to a procedure or function call that abstracts functionality or details not shown at this level. Can optionally be linked to another diagram for elaboration.



Else Block Represents an "else" block portion of a diagram block



**Message**

A simple message between header elements



**Return Message**

A return message between header elements



**Sequence diagram:**



Sequence Diagrams are used primarily to design, document and validate the architecture, interfaces and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario. UML sequence diagrams are useful design tools because they provide a dynamic view of the system behaviour which can be difficult to extract from static diagrams or specifications.

## **7.5. USE CASE DIAGRAM**

A use case diagram is a graph of actors set of use cases enclosed by a system boundary, communication associations between the actors and users and generalization among use cases. The use case model defines the outside (actors) and inside (use case) of the system’s behaviour.

Use case diagram is quite simple in nature and depicts two types of elements: one representing the business roles and the other representing the business processes.

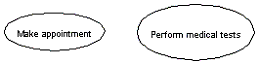


*Figure 3.1: an actor in a use case diagram*

To identify an actor, search in the problem statement for business terms that portray roles in the system. For example, in the statement "patients visit the doctor in the clinic for medical tests," "doctor" and "patients" are the business roles and can be easily identified as actors in the system.

**Use case:** A use case in a use case diagram is a visual representation of a distinct business functionality in a system. The key term here is "distinct business functionality." To choose a business process as a likely candidate for modelling as a use case, you need to ensure that the business process is discrete in nature.

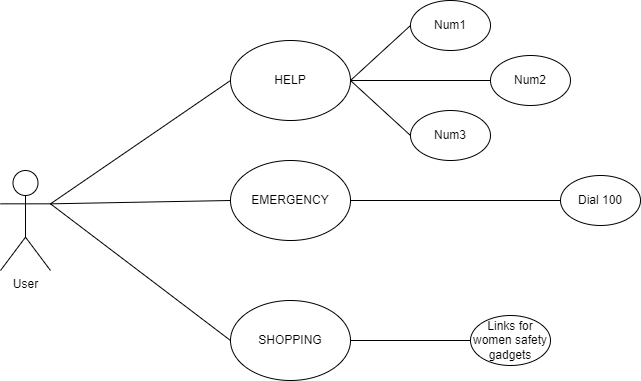
As the first step in identifying use cases, you should list the discrete business functions in your problem statement. Each of these business functions can be classified as a potential use case. Remember that identifying use cases is a discovery rather than a creation. As business functionality becomes clearer, the underlying use cases become more easily evident. An example of a use case is shown as an ellipse in a use case diagram (see Figure 3.2).



*Figure 3.2: use cases in a use case diagram*

Figure 3.2 shows two uses cases: "Make appointment" and "Perform medical tests" in the use case diagram of a clinic system. As another example, consider that a business process such as "manage patient records" can in turn have sub-processes like "manage patient's personal information" and "manage patient's medical information." Discovering such implicit use cases is possible only with a thorough understanding of all the business processes of the system through discussions with potential users of the system and relevant domain knowledge.

**USE CASE DIAGRAM:**



## **7.6. ACTIVITY DIAGRAM**

Activity diagrams represent the business and operational workflows of a system. An Activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state.

So, what is the importance of an Activity diagram, as opposed to a State diagram? A State diagram shows the different states an object is in during the lifecycle of its existence in the system, and the transitions in the states of the objects. These transitions depict the activities causing these transitions, shown by arrows.

An Activity diagram talks more about these transitions and activities causing the changes in the object states.

**Activity diagram:** 

**Defining an Activity diagram**

Let us take a look at the building blocks of an Activity diagram.

**Elements of an Activity diagram**

An Activity diagram consists of the following behavioural elements:

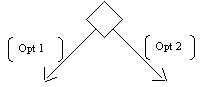
**Initial Activity:** This shows the starting point or first activity of the flow. Denoted by a solid circle. This is similar to the notation used for Initial State.



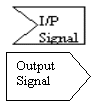
**Activity:** Represented by a rectangle with rounded (almost oval) edges.

.

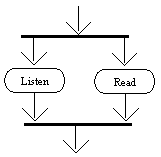
**Decisions:** Similar to flowcharts, a logic where a decision is to be made is depicted by a diamond, with the options written on either sides of the arrows emerging from the diamond, within box brackets.



**Signal:** When an activity sends or receives a message, that activity is called a signal. Signals are of two types: Input signal (Message receiving activity) shown by a concave polygon and Output signal (Message sending activity) shown by a convex polygon.



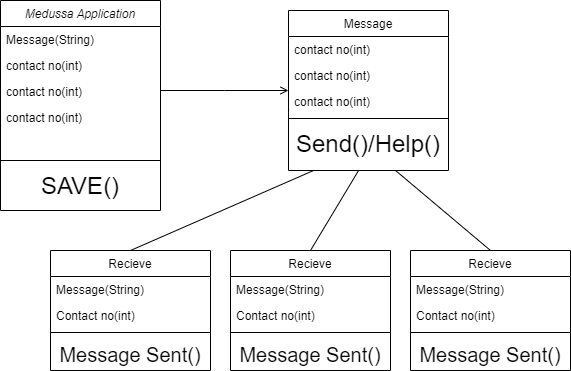
**Concurrent Activities:** Some activities occur simultaneously or in parallel. Such activities are called concurrent activities. For example, listening to the lecturer and looking at the blackboard is a parallel activity. This is represented by a horizontal split (thick dark line) and the two concurrent activities next to each other, and the horizontal line again to show the end of the parallel activity.



**Final Activity:** The end of the Activity diagram is shown by a bull's eye symbol, also called as a final activity.



## **7.7. CLASS DIAGRAM**

****

An object is any person, place, thing, concept, event, screen, or report applicable to your system. Objects both know things (they have attributes) and they do things (they have methods).

A class is a representation of an object and, in many ways, it is simply a template from which objects are created. Classes form the main building blocks of an object-oriented application.  Although thousands of students attend the university, you would only model one class, called *Student*, which would represent the entire collection of students.

**Responsibilities**

Classes are typically modelled as rectangles with three sections: the top section for the name of the class, the middle section for the attributes of the class, and the bottom section for the methods of the class. Attributes are the information stored about an object, while methods are the things an object or class do. For example, students have student numbers, names, addresses, and phone numbers. Those are all examples of the attributes of a student. Students also enrol in courses, drop courses, and request transcripts. Those are all examples of the things a student does, which get implemented (coded) as methods. You should think of methods as the object-oriented equivalent of functions and procedures.

## **7.8. STATE CHART DIAGRAM**

State chart diagram is used to describe the states of different objects in its life cycle. So the emphasis is given on the state changes upon some internal or external events. These states of objects are important to analyse and implement them accurately. State chart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

Before drawing a State chart diagram we must have clarified the following points:

* Identify important objects to be analysed.
* Identify the states.
* Identify the events.

**State diagram:**



## **7.9. DEPLOYEMENT DIAGRAM**

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed.

So deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

The purpose of deployment diagrams can be described as:

The name *Deployment* itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components where software components are deployed.

* Visualize hardware topology of a system.
* Describe the hardware components used to deploy software components.
* Describe runtime processing nodes.

# **8. FLOWCHART**

When we open the application, we can directly get the welcome screen then you need to register your personal details, contact no of recipients and a message. At that time your GPS location is on if it is off then you need to turn on the GPS. GPS will fetch your current location and when you click on save me button message will send to your recipients. And you can call to any emergency contact by clicking on it. Then you exit this application by clicking on exit button of mobile.

# **9. SAMPLE CODE**

## **9.1. MainActivity**

package com.coign.womensecurity;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.os.Handler;

import android.view.Menu;

import android.view.MenuItem;

public class MainActivity extends Activity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

        new Handler().postDelayed(new Runnable() {

            @Override

            public void run() {

                // TODO Auto-generated method stub

                startActivity(new Intent(MainActivity.this,SecondActivity.class));

            }

        } , 5000);

    }

}

## **9.2. SecondActivity**

package com.coign.womensecurity;

import java.util.List;

import java.util.Locale;

import android.app.Activity;

import android.content.Intent;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.location.Address;

import android.location.Geocoder;

import android.location.LocationManager;

import android.net.Uri;

import android.os.Bundle;

import android.telephony.SmsManager;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

import android.widget.ImageButton;

import android.widget.TextView;

import android.widget.Toast;

public class SecondActivity extends Activity {

Button help,save,display,clear;

LocationManager locationManager;

String provider;

TextView val;

GPSTracker gps;

SQLiteDatabase db;

String msg,con1,con2,con3,StateName,CityName,CountryName,add,premises,street,local;

double latitude,longitude;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_second);

        db=openOrCreateDatabase("coign", MODE\_PRIVATE, null);

        db.execSQL("create table if not exists map(lat double,lon double )");

        ImageButton btsettings=(ImageButton)findViewById(R.id.imageButton1);

        Button bthelp=(Button)findViewById(R.id.button1);

        db = openOrCreateDatabase("coign", MODE\_PRIVATE,null);

        btsettings.setOnClickListener(new OnClickListener() {

            @Override

            public void onClick(View arg0) {

                // TODO Auto-generated method stub

                Intent it=new Intent(SecondActivity.this,DetailsActivity.class);

                startActivity(it);

            }

        });

        bthelp.setOnClickListener(new OnClickListener() {

            @Override

            public void onClick(View arg0) {

                // TODO Auto-generated method stub

                Cursor c=db.rawQuery("select \* from contacts3", null);

                c.moveToFirst();

                if(c!=null)

                {

                    do

                    {

                        if(c.getCount()>0)

                        msg=c.getString(c.getColumnIndex("msg"));

                        con1=c.getString(c.getColumnIndex("name1"));

                        con2=c.getString(c.getColumnIndex("name2"));

                        con3=c.getString(c.getColumnIndex("name3"));

                        Toast.makeText(getApplicationContext(), "okkkkkkk", 100).show();

                        }while(c.moveToNext());

                }

             gps = new GPSTracker(SecondActivity.this);

                    if(gps.canGetLocation()){

                         latitude = gps.getLatitude();

                         longitude = gps.getLongitude();

                         Toast.makeText(getApplicationContext(), "Your Location is - \nLat: " + latitude + "\nLong: " + longitude, Toast.LENGTH\_SHORT).show();

                          getlocaddress();

                          String uri = "http://maps.google.com/maps?q=loc:" + latitude + "," + longitude + "";

                         sendmsg(uri);

                    }else{

                        gps.showSettingsAlert();

                    }

                    }

        private void getlocaddress() {

                // TODO Auto-generated method stub

            try

            {

                Toast.makeText(getApplicationContext(), "Your Location is - \nLat: " + latitude + "\nLong: in addressss" + longitude, Toast.LENGTH\_SHORT).show();

            Geocoder geocoder;

            List<Address> addresses;

            geocoder = new Geocoder(SecondActivity.this, Locale.getDefault());

            addresses = geocoder.getFromLocation(latitude, longitude, 1);

                street=addresses.get(0).getAddressLine(0);

               local=addresses.get(0).getSubAdminArea();

            StateName= addresses.get(0).getAdminArea();

            CityName = addresses.get(0).getLocality();

            CountryName = addresses.get(0).getCountryName();

            premises=addresses.get(0).getPremises();

            System.out.println(" StateName " + StateName);

            System.out.println(" CityName " + CityName);

            System.out.println(" CountryName " + CountryName);

            System.out.println(" CountryName " + premises);

            add=street+"\n"+local+"\n"+StateName+"\n"+CityName+"\n"+CountryName;

            Toast.makeText(getApplicationContext(), StateName+CityName+CountryName, Toast.LENGTH\_SHORT).show();

            }

            catch (Exception e)

            {

                e.printStackTrace();

            }

            }

        @SuppressWarnings("deprecation")

        private void sendmsg(String ss) {

            // TODO Auto-generated method stub

            /\*Intent intent = new Intent(Intent.ACTION\_VIEW, Uri.parse(uri));

            startActivity(intent);\*/

            SmsManager sm=SmsManager.getDefault();

            sm.sendTextMessage(con1, null, msg+"\n"+add, null, null);

            sm.sendTextMessage(con2, null, msg+"\n"+add, null,null);

            sm.sendTextMessage(con3, null,msg+"\n"+add, null,null);

            SmsManager sm2=SmsManager.getDefault();

            sm2.sendTextMessage(con1, null, msg+"\n"+ss, null, null);

            sm2.sendTextMessage(con2, null, msg+"\n"+ss, null,null);

            sm2.sendTextMessage(con3, null,msg+"\n"+ss, null,null);

        }

        });

    }

      @Override

      protected void onResume() {

        super.onResume();

      }

      @Override

      protected void onPause() {

        super.onPause();

      }

}

## **9.3. DetailsActivity**

package com.coign.womensecurity;

import android.app.Activity;

import android.content.ContentResolver;

import android.content.Intent;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.net.Uri;

import android.os.Bundle;

import android.provider.ContactsContract.CommonDataKinds.Phone;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

import android.widget.EditText;

import android.widget.Toast;

public class DetailsActivity extends Activity {

SQLiteDatabase db;

int i=100,j=200,k=300;

EditText et1,et2,et3,et4;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_details);

        Button btcon1=(Button)findViewById(R.id.button1);

        Button btcon2=(Button)findViewById(R.id.button2);

        Button btcon3=(Button)findViewById(R.id.button3);

        Button btsave=(Button)findViewById(R.id.button4);

         et1=(EditText)findViewById(R.id.editText1);

         et2=(EditText)findViewById(R.id.editText2);

         et3=(EditText)findViewById(R.id.editText3);

         et4=(EditText)findViewById(R.id.editText4);

         db=openOrCreateDatabase("coign", MODE\_PRIVATE, null);

         db.execSQL("create table if not exists contacts3(name1 varchar(20),name2 varchar(20),name3 varchar(20),msg varchar(100))");

         getdetails();

         btcon1.setOnClickListener(new OnClickListener() {

            @Override

            public void onClick(View arg0) {

                // TODO Auto-generated method stub

                String s1=et1.getText().toString();

                Intent it=new Intent(Intent.ACTION\_PICK,Uri.parse("content://contacts"));

                it.setType(Phone.CONTENT\_TYPE);

                startActivityForResult(it, i);

            }

        });

         btcon2.setOnClickListener(new OnClickListener() {

            @Override

            public void onClick(View arg0) {

                // TODO Auto-generated method stub

                String s2=et2.getText().toString();

                Intent it=new Intent(Intent.ACTION\_PICK,Uri.parse("content://contacts"));

                it.setType(Phone.CONTENT\_TYPE);

                startActivityForResult(it, j);

            }

        });

         btcon3.setOnClickListener(new OnClickListener() {

            @Override

            public void onClick(View arg0) {

                // TODO Auto-generated method stub

                String s3=et3.getText().toString();

                Intent it=new Intent(Intent.ACTION\_PICK,Uri.parse("content://contacts"));

                it.setType(Phone.CONTENT\_TYPE);

                startActivityForResult(it, k);

            }

        });

         btsave.setOnClickListener(new OnClickListener() {

            @Override

            public void onClick(View arg0) {

                // TODO Auto-generated method stub

                String s1=et1.getText().toString();

                String s2=et2.getText().toString();

                String s3=et3.getText().toString();

                String s4=et4.getText().toString();

                db.execSQL("insert into contacts3 values('"+s1+"','"+s2+"','"+s3+"','"+s4+"')");

                Toast.makeText(getApplicationContext(), "Data Saved", Toast.LENGTH\_LONG).show();

            }

        });

    }

    private void getdetails() {

        // TODO Auto-generated method stub

        Cursor c=db.rawQuery("select \* from contacts3", null);

        c.moveToFirst();

        if(c!=null)

        {

            do

            {

                if(c.getCount()>0)

                {

            int i=c.getColumnIndex("name1");

            String scon1=c.getString(i);

            int j=c.getColumnIndex("name2");

            String scon2=c.getString(j);

            int k=c.getColumnIndex("name3");

            String scon3=c.getString(k);

            int l=c.getColumnIndex("msg");

            String smsg=c.getString(l);

            et1.setText(scon1);

            et2.setText(scon2);

            et3.setText(scon3);

            et4.setText(smsg);

                }

            }

            while(c.moveToNext());

        }

    }

    @Override

        protected void onActivityResult(int requestCode, int resultCode, Intent data) {

            // TODO Auto-generated method stub

            super.onActivityResult(requestCode, resultCode, data);

            if (requestCode==i) {

                if (resultCode==RESULT\_OK) {

                    Uri uri=data.getData();

                    ContentResolver cr=getContentResolver();

                    String[] OBJ={

                            Phone.NUMBER

                    };

                    Cursor c=cr.query(uri, OBJ, null, null, null);

                    c.moveToFirst();

                    if (c.getCount()>0) {

                        int i=c.getColumnIndex(Phone.NUMBER);

                        String s1=c.getString(i);

                        et1.setText(s1);

                    }

                }

            }

            else if (requestCode==j) {

                if (resultCode==RESULT\_OK) {

                    Uri uri=data.getData();

                    ContentResolver cr=getContentResolver();

                    String[] OBJ={

                            Phone.NUMBER

                    };

                    Cursor c=cr.query(uri, OBJ, null, null, null);

                    c.moveToFirst();

                    if (c.getCount()>0) {

                        int j=c.getColumnIndex(Phone.NUMBER);

                        String s2=c.getString(j);

                        et2.setText(s2);

                    }

                }

            }

            else if (requestCode==k) {

                if (resultCode==RESULT\_OK) {

                    Uri uri=data.getData();

                    ContentResolver cr=getContentResolver();

                    String[] OBJ={

                            Phone.NUMBER

                    };

                    Cursor c=cr.query(uri, OBJ, null, null, null);

                    c.moveToFirst();

                    if (c.getCount()>0) {

                        int k=c.getColumnIndex(Phone.NUMBER);

                        String s3=c.getString(k);

                        et3.setText(s3);

                    }

                }

            }

        }

}

## **9.4. IncomingCall**

package com.coign.womensecurity;

import java.util.List;

import java.util.Locale;

import android.content.BroadcastReceiver;

import android.content.Context;

import android.content.Intent;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.location.Address;

import android.location.Geocoder;

import android.location.LocationManager;

import android.telephony.PhoneStateListener;

import android.telephony.SmsManager;

import android.telephony.TelephonyManager;

import android.widget.Toast;

public class Incomingcall extends BroadcastReceiver{

SQLiteDatabase db;

LocationManager locationManager;

GPSTracker gps;

String msg,con1,con2,con3,StateName,CityName,CountryName,add,premises,street,local;

double latitude,longitude;

    String incomingNumber,text;

     public static String missedcalnumber,ph;

     boolean ringing = false;

        boolean offhook = false;

    @Override

    public void onReceive( final Context arg0, Intent arg1) {

        // TODO Auto-generated method stub

         try{

               TelephonyManager telephony = (TelephonyManager)arg0.getSystemService(Context.TELEPHONY\_SERVICE);

               telephony.listen(new PhoneStateListener(){

                   @Override

                   public void onCallStateChanged(int state, String incomingNumber) {

                     super.onCallStateChanged(state, incomingNumber);

                    db=arg0.openOrCreateDatabase("coign", arg0.MODE\_PRIVATE, null);

                     switch(state){

                        case TelephonyManager.CALL\_STATE\_RINGING:

                            ringing = true;

                            offhook = false;

                        break;

                        case TelephonyManager.CALL\_STATE\_OFFHOOK:

                            offhook = true;

                            ringing = false;

                              break;

                        case TelephonyManager.CALL\_STATE\_IDLE:

                                  ringing = false;

                                  offhook = false;

                              missedcalnumber=incomingNumber;

                             // missedcalnumber=missedcalnumber.replaceAll("[\\D] ","");

                              missedcalnumber=missedcalnumber.replaceAll(" ","");

                              Toast.makeText(arg0, "you have a missed call:"+ missedcalnumber, Toast.LENGTH\_LONG).show();

                            try {

                              Cursor c = db.rawQuery("select \* from contacts3", null);

                                if(c!=null)

                                {

                                    c.moveToFirst();

                                    {

                                        do {

                                            int i=c.getColumnIndex("name1");

                                            con1=c.getString(i);

                                            int j=c.getColumnIndex("name2");

                                            con2=c.getString(j);

                                            int k=c.getColumnIndex("name3");

                                            con3=c.getString(k);

                                        /\*  int l=c.getColumnIndex("msg");

                                            String smsg=c.getString(l);\*/

                                            Toast.makeText(arg0,con1+"  aaaa  "+con2+" "+con3, 100).show();

                                              if(missedcalnumber.length()==13)

                                                {

                                                ph=missedcalnumber.substring(3);

                                                Toast.makeText(arg0, "Sstring"+ph, Toast.LENGTH\_LONG).show();

                                      //   String  number=c.getString(c.getColumnIndex("phnumber"));

                                            if( ph.equals(con1)|| ph.equals(con2)|| ph.equals(con3) )

                                            {

                                                gps = new GPSTracker(arg0);

                                                double lat=gps.latitude;

                                                double lon = gps.longitude;

                                                 text = "Latitude:"+lat+","+"Longitude:"+lon;

                                                  String uri = "http://maps.google.com/maps?q=loc:" +lat + "," + lon;

                                               SmsManager smsManager = SmsManager.getDefault();

                                          smsManager.sendTextMessage(missedcalnumber, null, uri, null, null);

                                       Toast.makeText(arg0, "SMS Sent!", Toast.LENGTH\_LONG).show();

                                            }

                                          } }

                                       while (c.moveToNext());

                                    }

                                }

                            }catch (Exception e) {

                                Toast.makeText(arg0,

                                        "SMS failed, please try again later!",

                                        Toast.LENGTH\_LONG).show();

                                e.printStackTrace();

                            }

                             break;

                        default:

                break;

                   }

                   }

               },PhoneStateListener.LISTEN\_CALL\_STATE);

               db.close();

             }

             catch (Exception e) {

                    e.printStackTrace();

                }

           }

}

## **9.5. GPSTracker**

package com.coign.womensecurity;

import org.apache.http.util.LangUtils;

import android.app.AlertDialog;

import android.app.Service;

import android.content.Context;

import android.content.DialogInterface;

import android.content.Intent;

import android.location.Location;

import android.location.LocationListener;

import android.location.LocationManager;

import android.os.Bundle;

import android.os.IBinder;

import android.provider.Settings;

import android.util.Log;

import android.widget.Toast;

public class GPSTracker extends Service implements LocationListener {

    private final Context mContext;

    // flag for GPS status

    boolean isGPSEnabled = false;

    // flag for network status

    boolean isNetworkEnabled = false;

    // flag for GPS status

    boolean canGetLocation = false;

    Location location; // location

    double latitude; // latitude

    double longitude; // longitude

    // The minimum distance to change Updates in meters

    private static final long MIN\_DISTANCE\_CHANGE\_FOR\_UPDATES = 10; // 10 meters

    // The minimum time between updates in milliseconds

    private static final long MIN\_TIME\_BW\_UPDATES = 1000 \* 60 \* 1; // 1 minute

    // Declaring a Location Manager

    protected LocationManager locationManager;

    public GPSTracker(Context context) {

        this.mContext = context;

        getLocation();

    }

    public Location getLocation() {

        try {

            locationManager = (LocationManager) mContext

                    .getSystemService(LOCATION\_SERVICE);

            // getting GPS status

            isGPSEnabled = locationManager

                    .isProviderEnabled(LocationManager.GPS\_PROVIDER);

            // getting network status

            isNetworkEnabled = locationManager

                    .isProviderEnabled(LocationManager.NETWORK\_PROVIDER);

            if (!isGPSEnabled && !isNetworkEnabled) {

                // no network provider is enabled

            } else {

                this.canGetLocation = true;

                if (isNetworkEnabled) {

                    locationManager.requestLocationUpdates(

                            LocationManager.NETWORK\_PROVIDER,

                            MIN\_TIME\_BW\_UPDATES,

                            MIN\_DISTANCE\_CHANGE\_FOR\_UPDATES, this);

                    Log.d("Network", "Network");

                    if (locationManager != null) {

                        location = locationManager

                                .getLastKnownLocation(LocationManager.NETWORK\_PROVIDER);

                        if (location != null) {

                            latitude = location.getLatitude();

                            longitude = location.getLongitude();

                        }

                    }

                }

                // if GPS Enabled get lat/long using GPS Services

                if (isGPSEnabled) {

                    if (location == null) {

                        locationManager.requestLocationUpdates(

                                LocationManager.GPS\_PROVIDER,

                                MIN\_TIME\_BW\_UPDATES,

                                MIN\_DISTANCE\_CHANGE\_FOR\_UPDATES, this);

                        Log.d("GPS Enabled", "GPS Enabled");

                        if (locationManager != null) {

                            location = locationManager

                                    .getLastKnownLocation(LocationManager.GPS\_PROVIDER);

                            if (location != null) {

                                latitude = location.getLatitude();

                                longitude = location.getLongitude();

                            }

                        }

                    }

                }

            }

        } catch (Exception e) {

            e.printStackTrace();

        }

        return location;

    }

    /\*\*

     \* Stop using GPS listener

     \* Calling this function will stop using GPS in your app

     \* \*/

    public void stopUsingGPS(){

        if(locationManager != null){

            locationManager.removeUpdates(GPSTracker.this);

        }

    }

    /\*\*

     \* Function to get latitude

     \* \*/

    public double getLatitude(){

        if(location != null){

            latitude = location.getLatitude();

        }

        // return latitude

        return latitude;

    }

    /\*\*

     \* Function to get longitude

     \* \*/

    public double getLongitude(){

        if(location != null){

            longitude = location.getLongitude();

        }

        // return longitude

        return longitude;

    }

    /\*\*

     \* Function to check GPS/wifi enabled

     \* @return boolean

     \* \*/

    public boolean canGetLocation() {

        return this.canGetLocation;

    }

    /\*\*

     \* Function to show settings alert dialog

     \* On pressing Settings button will lauch Settings Options

     \* \*/

    public void showSettingsAlert(){

        AlertDialog.Builder alertDialog = new AlertDialog.Builder(mContext);

        // Setting Dialog Title

        alertDialog.setTitle("GPS is settings");

        // Setting Dialog Message

        alertDialog.setMessage("GPS is not enabled. Do you want to go to settings menu?");

        // On pressing Settings button

        alertDialog.setPositiveButton("Settings", new DialogInterface.OnClickListener() {

            public void onClick(DialogInterface dialog,int which) {

                Intent intent = new Intent(Settings.ACTION\_LOCATION\_SOURCE\_SETTINGS);

                mContext.startActivity(intent);

            }

        });

        // on pressing cancel button

        alertDialog.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {

            public void onClick(DialogInterface dialog, int which) {

            dialog.cancel();

            }

        });

        // Showing Alert Message

        alertDialog.show();

    }

    @Override

    public void onLocationChanged(Location location) {

    }

    @Override

    public void onProviderDisabled(String provider) {

    }

    @Override

    public void onProviderEnabled(String provider) {

    }

    @Override

    public void onStatusChanged(String provider, int status, Bundle extras) {

    }

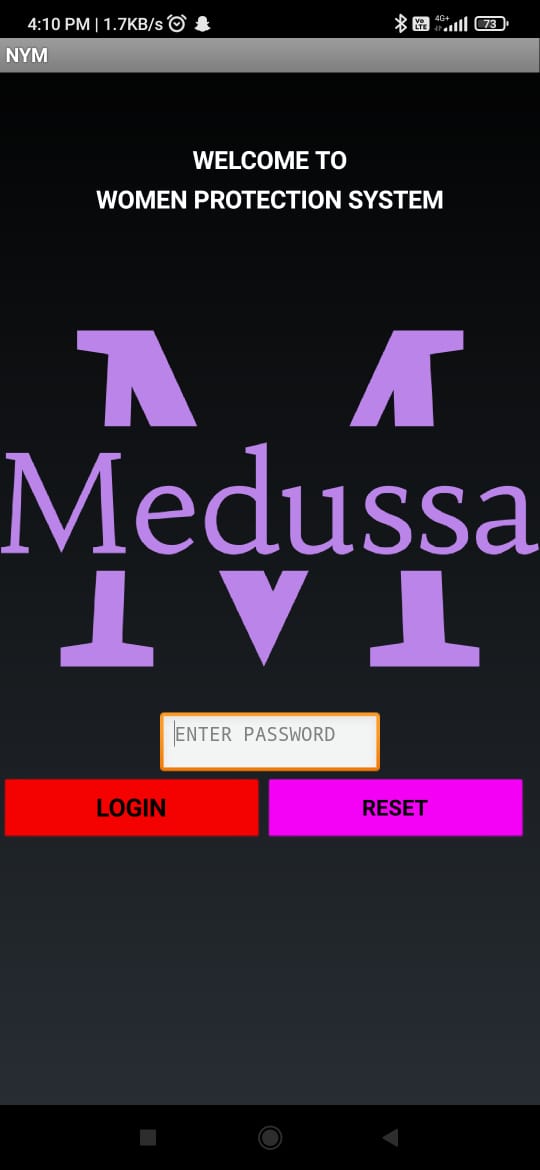
    @Override

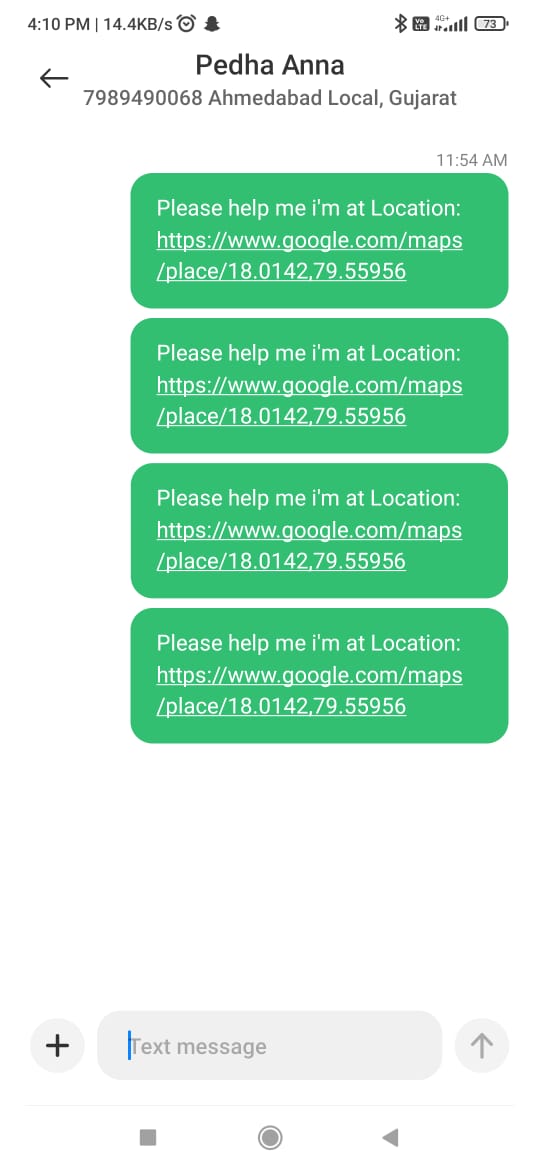
    public IBinder onBind(Intent arg0) {

        return null;

    }

**10. SAMPLE SCREEN**

****

****

# **11. FUTURE ENHANCEMENTS**

A few potential future enhancements for women safety applications:

1. Integration with wearable technology: Women safety applications could integrate with wearable technology such as smartwatches or fitness trackers to provide real-time tracking and alerts in case of an emergency.

2. Artificial intelligence-powered threat detection: Advanced AI algorithms can analyze patterns of behavior and identify potential threats, such as following someone or loitering around an area. This can help in preventing incidents before they occur.

3. Improved accuracy of location tracking: Location tracking technology can be improved to provide more precise location data in emergency situations, which can help authorities respond more quickly.

4. Voice-activated distress signals: In situations where it may be difficult to access a phone, voice-activated distress signals could be used to trigger an alarm and notify emergency services.

5. Integration with public safety databases: Women safety applications can be integrated with public safety databases to provide users with real-time information about crime in their area and potential threats.

6. Emergency response coordination: Women safety applications can also be integrated with emergency response services to coordinate a response in real-time.

Overall, these enhancements can improve the effectiveness of women safety applications and help to create a safer environment for women.

# **12. METHOD OF IMPLEMENTATION**

Implementation literally means to put into effect or to carry out. The system implementation phase of the software deals with the translation of the design specifications into the source code. The ultimate goal of the implementation is to write the source code and the internal documentation so that it can be verified easily. The code and documentation should be written in a manner that eases debugging, testing and modification. System flowcharts, sample run on packages, sample output etc. Is part of the implementation?

An effort was made to satisfy the following goals in order specified.

• Minimization of Response Time.

• Clarity and Simplicity of the Code.

• Minimization of Hard-Coding.

Various types of bugs were discovered while debugging the modules. These ranged from logical errors to failure on account of various processing cases.

# **13. TESTING AND VALIDATION**

The development of software systems involves of a series of production activities where opportunities for injection of human fallibilities are enormous. Errors may begin to occur at the very inception of the process where the objectives may be erroneously or imperfectly specified, as well as in later design and development stages. Because of human inability to perform and communicate with perfection, software development is accompanied by a quality assurance activity.

## **13.1. TESTING TECHNIQUES**

Testing is the process of executing a program with the intention of finding errors. The various test strategies used for testing the software are as follows.

### **13.1.1 Unit Testing**

Unit testing focuses on verification effort on the smallest unit of the software design module. The main goal is to make sure that every source statement and logic path has been executed correctly at least once. The output of this stage is the source code.

### **13.1.2 Integration Testing**

In Integration testing, we find errors that have occurred during the integration. After testing each module, which is then integrated into subsystems and then to form the entire system on which integration testing is performed. The goal of testing is to detect the design errors, while focusing on the testing the interconnection between modules.

### **13.1.3 Validation Testing**

This testing concentrates on confirming that the software is error-free in all respects. All the specified validations are verified and the software is subjected to hard-core testing. It also aims at determining the degree of deviation that exists in the software designed from the specification; they are listed out and are corrected.

### **13.1.4 System Testing**

In this testing, the system is tested for the errors after coupling all the modules together The system is tested against the specified requirements to see if all the requirements are met and the system performs as specified by the requirements.

# **14. CONCLUSION**

In this study, we proposed the design and implementation of a women's safety system in the form of an application. A location tracking subsystem was successfully built in accordance with the objectives, and the necessary findings were reported. The system will be expanded in accordance with the goals outlined in the future scope.

The study also discusses GPS technology, which may be used to monitor the victim's whereabouts using latitudes and longitudes.

To summarize, our software provides a safe and secure environment for women in society, allowing them to work late into the evening. Anyone thinking about committing a crime against a woman will be stopped, and the rate of crime against women will drop. This program will act as a weapon for women, safeguarding their safety and security, and it will run on any Android smartphone. With additional research and development, our concept might be implemented on a small wearable device like a watch, necklace, or bracelet that will be made with GPS and GSM modules. When this system is engaged, the GPS module collects GPS data and encodes it into a valid Google maps link, which is subsequently texted to the recruited family and friends.

# **15. BIBILOGRAPHY**

## **15.1. REFERENCES**

* Herbert Schildt.2008 ,”Java Complete Reference”, Tata McGraw-Hill ,

7th Edition, pp.177-180 .

* Grady Booch, James Rambaugh.1998,“UnifiedModeling Language User Guide”, Addison Wesley Publishing, chapter 8-31.

## **15.2. REFERRED URLS**

[www.android.com](http://www.android.com/)

<http://developer.android.com/index.html>

<http://en.wikipedia.org/wiki/SQLite>

<https://timesofindia.indiatimes.com/7-best-women-safety-apps/photostory/51285556.cms>

<https://womensafetywing.telangana.gov.in/women-safety-apps/>

<https://www.researchgate.net/publication/352386903_Android_App_for_Women_Safety>